AD-A104 524   NAVAL RESEARCH LAB   WASHINGTON DC   F/G 9/2
ARCHITECTURE TUNING OF A REAL-TIME SIGNAL SORTER IN A DENSE ENV--ETC(U)
SEP 81   T R HUSSON, R H EVANS, L H MOFFETT
UNCLASSIFIED   NRL-8495   NL

1 OF 1
AD A
04524

END
DATE
FILMED
10-81
DTIC

NRL Report 8495

Architecture Tuning of a Real-Time Signal Sorter in a Dense Environment

T. R. Husson, R. H. Evans, and L. H. Moffett

Advanced Techniques Branch
Tactical Electronic Warfare Division

A. M. Abdalla

Locus Inc.

September 11, 1981

NAVAL RESEARCH LABORATORY
Washington, D.C.

Approved for public release; distribution unlimited.

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER NRL Report 8495	2. GOVT ACCESSION NO. AD-A104524	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) ARCHITECTURE TUNING OF A REAL-TIME SIGNAL SORTER IN A DENSE ENVIRONMENT		5. TYPE OF REPORT & PERIOD COVERED (7) Interim rep~~ort on a continuing NRL problem~~
		6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s) (10) T. R. Husson, R. H. Evans, L. H. Moffett, ~~and~~ A. M. Abdalla*		8. CONTRACT OR GRANT NUMBER(s)
9. PERFORMING ORGANIZATION NAME AND ADDRESS Naval Research Laboratory Washington, DC 20375		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS 62734N; 0534-0
11. CONTROLLING OFFICE NAME AND ADDRESS Naval Research Laboratory Washington, DC 20375	(11) 11	12. REPORT DATE Sept~~ember 11, 19~~81
		13. NUMBER OF PAGES 33
14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office)		15. SECURITY CLASS. (of this report) UNCLASSIFIED
		15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

*Locus, Inc.

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

Computer architecture tuning
Signal processing
Signal sorting
Microprogramming
Associative memory

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

This report describes the results of a study of the architecture of a real-time signal sorting system. The purpose of the signal sorter is to process and identify the outputs of a radar receiver. The problems of high data rates and uncertain environment characteristics require an adaptive system. The study began with a previously designed signal sorting architecture and the effects of various dynamic architecture tuning schemes on that system were analyzed.

CONTENTS

ARCHITECTURE TUNING OF A REAL-TIME SIGNAL SORTER IN A DENSE ENVIRONMENT

INTRODUCTION

Signal sorting involves the correlation of the mass of signals detected by a receiver with the individual sources that generate each signal. The purpose of the receiver is to capture and measure the characteristics of the various signals that comprise the electromagnetic environment. The receiver attempts to separate the individual pulses and determine their parameters. These measured parameters include the direction of arrival, carrier frequency and pulse width. These measured values for each pulse seen by the receiver are expressed in digital form and are the outputs of the receiver. The outputs are provided in real-time to the signal sorter. The signal sorter receives this pulse train in which the individual pulses might belong to many different emitter sources. Based on measurements on the individual pulses, the signal sorter should recognize the emitters present in the environment and separate the interleaved signal into the individual sources.

This separation and identification of emitters is extremely important in a threat environment. The needs in such an environment are to separate the friendly emitters from the unfriendly emitters and also to determine the number and type of unfriendly emitters. Such threat environments are usually characterized by a large concentration of electromagnetic activity, making the real-time signal sorting problem even more difficult.

Before a detected signal may be correlated with a particular emitter, a file on that emitter must first be established by the signal sorter. These files are generated by the signal sorter using measured parameters such as carrier frequency (CF), pulse width (PW), direction of arrivals (DOA), latest time of arrival (TOA), and computed parameters such as pulse repetition interval (PRI).

Many irregularities could be present in the set of signals detected by the signal sorting system making the sorting task much more difficult. Signal drop out, signal overlap, measurement inconsistencies, and intentional variation of the signal parameters by the emitter may all contribute to the variations seen by the signal sorter. If these irregularities are severe, extra hardware and/or software must be added to alleviate the effects of the signal variations. This research has focused only on the problem of high data rates caused by the dense environment. Current research is being performed on the added problems of signal irregularities, such as frequency agile emitters, to the tasks of signal sorting. This work will be documented in future reports.

The high data rate compounds the signal sorting problem by requiring high throughput rates in order to achieve the real time requirements. Since airborne platforms are to use the signal sorting system, a brute force approach using a single large powerful computer is not feasible. Efficient use of smaller microprocessors is needed to keep up with the high data rate and the real time processing requirements.

A software model for a signal sorting system to handle greater than a million pulses per second of radar signals, in a dense environment has been developed. A simulation of the environment and a hardware model of the signal sorter have also been developed. Based on the results of the simulation, the hardware model was constructed.

Manuscript submitted May 8, 1981.

The purpose of this research is to investigate the application of dynamic architecture tuning to the signal sorter system in order to improve performance. Several techniques have been investigated and reported in this report. Test runs have been made for these different architectures using the software simulation.

The term tuning implies that the computer system structure is adjusted to solve the given application more efficiently. Dynamic tuning implies that changes in the computer system structure could occur during operation based on real-time measurements on the state of the system. Using algorithms determined through simulation, the sorter dynamically reconfigures itself in a true feedback control manner to optimize its performance under the various signal environments.

A brief description of the environment simulation and the computing system which has been built are given in the next few sections. This is followed by the investigation of dynamic tuning and its effect on system performance.

ENVIRONMENT SIMULATION

In order to design and simulate a signal sorting system, a software simulation model of the environment was first constructed. This model is capable of generating interleaved pulse trains representing several types of emitters such as would be encountered in a real environment. *Regular* emitters which do not have intentional variations of their parameters are considered to be the predominant type of emitter seen. Exotic emitters which intentionally vary their *PRI* or *Carrier Frequency* on a pulse to pulse basis are also represented in the environment simulation. Another class of emitters which is also included in the environment model is the *Pulse Group* in which groups of pulses rather than a single pulse are transmitted *at every pulse group interval.*

The parameters associated with each emitter such as carrier frequency, PRI, emitter location, etc. are randomly selected from a uniform distribution with limits representing a practical range of values. A variation in some of the parameters is added on a pulse to pulse basis to simulate any measurement irregularities, emitter agility and emitter drift. These variations are also selected from uniform distributions which represent the expected range of values. Sufficient flexibility is included in the model to permit the selection of various signal densities in different runs and to allow the same environment to be tested against several sorter designs.

Simulation of the electromagnetic activity seen by the receiver is done in the DAGE and EMIT subroutines; a listing for each is included in the appendix. The combination of these routines can provide various environments in terms of the number of emitters seen, the location of these emitters with respect to the receiver, and the parameters associated with the electromagnetic transmissions from each emitter. It is also possible to use the same environment in any number of runs. This capability allows different signal sorter architectures to be compared with a common data base and also permits the parameters associated with a single architecture to be tuned for optimal performance.

Definition of the parameters associated with each emitter is performed by the DAGE routine. Values for each parameter are randomly selected by DAGE from a distribution of expected values for that parameter. A list of the parameter array defined for each emitter is shown in Fig. 1. After DAGE initializes values for all parameters in every emitter, control is passed to the EMIT routine.

Time varying parameters for each emitter are modified and updated on a pulse-to-pulse basis in the EMIT routine. Calculation of both the emitter main beam power level at the receiving antenna and the time of arrival (TOA) of the transmitted pulse at the receiving antenna are done in the EMIT routine.

E(I,J) WHERE I = EMITTER NUMBER
J = DEFINED BELOW

1. Dx - x DISPLACEMENT IN METERS
2. Dy - y DISPLACEMENT IN METERS
3. POWER OF EMITTER (INCLUDES ANTENNA GAIN)
4. MAINLOBE SIZE - OF EMITTER IN DEGREES
5. SIDELOBE LOSS - IN dBs DOWN DROM MAIN BEAM GAIN
6. MAX. ANTENNA ANGLE - UPPER SCAN LIMIT OF EMITTER ANTENNA IN DEGREES
7. SCAN RATE - IN Hz
8. PRI - PULSE REPETITION INTERVAL OF EMITTER IN SECONDS
9. PULSE WIDTH - IN SECONDS
10. FREQUENCY - IN GHz
11. ON TIME - TIME AT WHICH THE EMITTER IS TURNED ON
12. OFF TIME - TIME AT WHICH THE EMITTER IS TURNED OFF
13. RCVR POWER - POWER LEVEL FROM EMITTER SEEN AT THE RECEIVER ANTENNA
14. TOA-TIME OF ARRIVAL OF TRANSMITTED PULSE AT THE RECEIVED ANTENNA
15. DOA - DIRECTION OF ARRIVAL OF TRANSMITTED PULSE AT THE RECEIVER ANTENNA
16. FLAG - SET FOR DURATION OF PULSE
17. INITIAL ANTENNA ANGLE - INITIAL POSITION OF EMITTER ANTENNA IN DEGREES
18. Dz - z DISPLACEMENT IN METERS
19. MIN. ANTENNA ANGLE - LOWER SCAN LIMIT OF EMITTER ANTENNA IN DEGREES
20. Vx - x VELOCITY COMPONENT OF EMITTER PLATFORM
21. Vy - y VELOCITY COMPONENT OF EMITTER PLATFORM
22. Vz - z VELOCITY COMPONENT OF EMITTER PLATFORM
23. TYPE: −1-MOVING EMITTER, 0-FIXED, +1-COLLISION COURSE
24. SPARE
25. SPARE

Fig. 1 — Field definition of the parameter array for each emitter

A model of a receiver and antenna system was also developed in the signal sorter design process. The philosophy of the antenna/receiver system was to model a feasible system that could be effectively used by a signal sorting system. The antenna system which is more suitable for this application is an array antenna with a beam forming network. The receiver simulation consists of 16 crystal video detectors, one at each beam port of the antenna system. Values for the sensitivity, bandwidth, gains, etc., were chosen to be well within the limits of current technology. The ability to measure the direction of arrival on a pulse to pulse basis was modeled in the antenna/receiver system, as this parameter is very useful in the sorting function of the system. Amplitude measurements from the crystal detectors at each beam are used to establish direction of arrival information. All the measured values are then digitized before being passed to the processing system for analysis and sorting.

CURRENT SYSTEM CONFIGURATION

A block diagram of the hardware configuration of the signal sorting system is shown in Fig. 2. It combines both parallel and pipeline architectures. Various stages of the pipeline structure are separated by first-in-first-out (FIFO) buffers to synchronize the rate of data flow.

In implementing the proposed architecture, the signal sorting task is divided into several subtasks: presort, identification and file generation, list forming of expected arrivals, and *content addressable memory* (CAM) load. The presort consists of a CAM whose contents are continuously

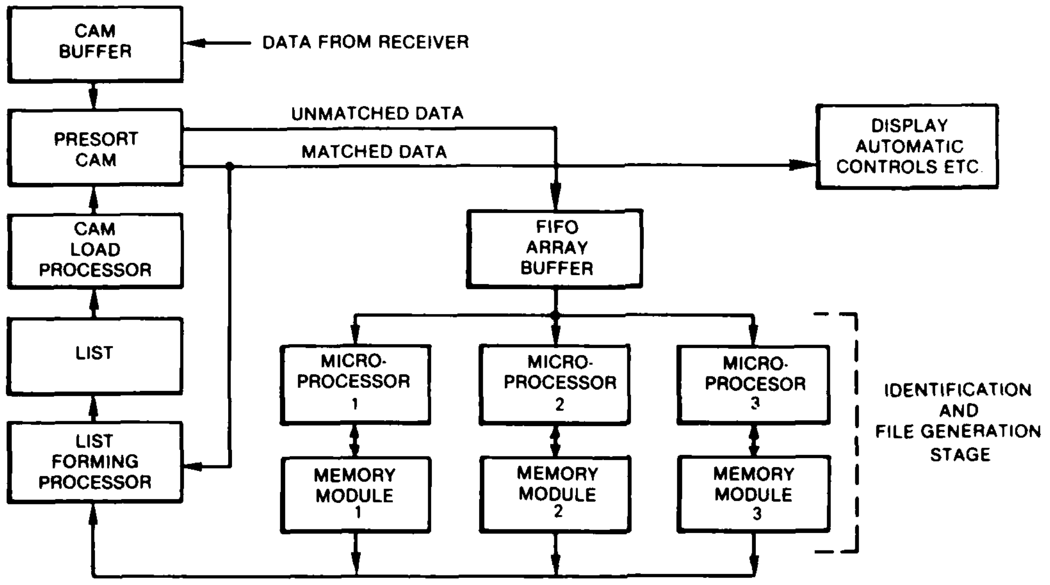


Fig. 2 — Signal sorting system

updated based on the known active emitters. The received data are compared in the CAM for a MATCH/NO MATCH indication. This stage is used as a filtering to the data stream and therefore reduces the rate of the data flow into the remaining stages of the pipeline. The identification and file generation stage receives only the unmatched data and correlates this to the file of previously identified emitters. The result of processing the unmatched data is either the addition of new emitters to the file or updating the file entry of a known emitter.

The LIST Forming Processor

The LIST consists of a number of FIFO buffers which are ordered as a sequence of time slots. Each buffer is loaded with those emitters whose expected next arrival times fall within the same time slot. The contents of the FIFOs comprising the LIST are loaded into the CAM one emitter at a time. No ordering is done on the data within a given FIFO, other than the built-in first-in-first-out characteristics of the buffer. Figure 3 illustrates the LIST and the CAM loading.

An emitter is loaded into the FIFO buffer corresponding to its *next time of arrival* (NTOA). The association of a FIFO buffer with NTOA is determined by the value of some middle bits of the NTOA referred to as "time slot" or "time window" bits. In order to have a uniform distribution of the emitters in the FIFO buffers, and to be able to load the emitter parameters into the CAM to provide the highest hit ratio, the time slot should be a function of the distribution of the PRI's of the emitters.

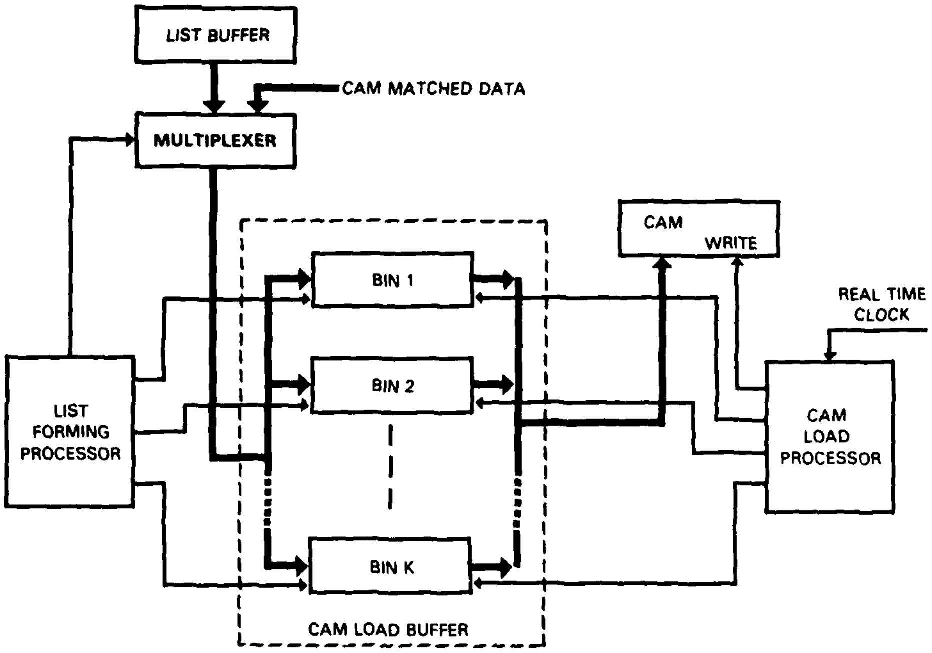


Fig. 3 — LIST forming/load CAM stages

If we assume a uniform distribution of PRIs over some known range of values for an environment, then for given values of the highest and the lowest PRI of the emitters in this environment, a time window for each FIFO could be selected based on the following:

NTOA

	M	N

let M = # of middle bits used as a time slot $(M > 0)$

N = # of bits to the right of the time slot in the NTOA work $(N > 0)$. This is the number of bits the NTOA word must be shifted to position the M "time slot" bits as the least significant bits of the word.

then M and N should satisfy

$$(1)\quad 2^{N} \leqslant \textit{Minimum PRI}$$

$$(2)\quad 2^{N+M} \geqslant \text{Maximum PRI}$$

where the value of the least significant bit (LSB) of the PRI word has the same weight as the LSB of the NTOA word and the real-time clock. The value of the LSB in these simulation results is 1 μs.

The value of M determines the number of FIFO buffers to be used. That is, the number of FIFO buffers in the LIST will be 2^M. In order to minimize the hardware, M should be kept to a minimum value which satisfies relations (1) and (2). Therefore these relations could be rewritten as:

$$2^N \leqslant PRI_L \qquad \text{or } N \leqslant [\text{Log}_2\, PRI_L]$$

$$2^{N+M} \geqslant PRI_H \qquad \text{or } N + M = [\text{Log}_2\, PRI_H] + 1$$

for PRI_L and PRI_H in μs,

where [ ] indicates the integer value.

As an example let $PRI_L = 300$ μs, $PRI_H = 2000$ μs.

The constraints (1) and (2) show:

$$N \leqslant 8 \qquad \text{or } 2^8 = 256 < 300$$

$$N + M = 11 \text{ or } 2^{11} = 2048 > 2000$$

(3,8), (4,7), (5,6), (6,5) are the (M,N) values which satisfy the problem constraints. Note that the (M,N) pairs (7,4), (8,3), (9,2), (10,1), (11,0) also satisfy the constraints. However, the practical constraints of the buffer depths and limiting the total amount of hardware make any of these solutions impractical. In order to minimize the hardware, the values of (M,N) should be taken as (3,8) which requires eight FIFO bins and an eight bit shift on the NTOA words (256 μs/bin).

Load CAM Processor

The function of the LIST Forming Processor is to form an ordered list of those emitters whose pulses are next expected to arrive. The Load CAM Processor then loads these emitters into the CAM shortly before their expected arrivals. Therefore, the list loading and the CAM loading are performed by the LIST Forming Processor and Load CAM Processor respectively as shown in Fig. 3. These two functions could possibly be performed by a single physical processor.

There are two sources for the data to be loaded into the LIST. The first source is the Associative Processor Emitter pulses which are matched in the CAM and are expected to arrive again delayed by a time equal to the emitter's PRI (Fig. 4). The associative processor has a PRI memory which stores the PRI's corresponding to all emitters which reside in the CAM. When a CAM match occurs, the expected next time of arrival (NTOA) of the matched emitter is computed (by adding its PRI to its TOA), and the emitter's data are loaded into the ordered LIST. The method of ordering the LIST when loading it, and the order of loading the CAM from the LIST are discussed later in this section.

The second source for the data to be loaded into the LIST is the microprocessor array. Emitters which are identified by the microprocessor array are loaded in the LIST Buffer (see Fig. 2). The Load CAM processor then reads the emitter parameters from the buffer and loads them into the LIST.

The load CAM processor loads the CAM from the LIST at those times when the CAM is not busy processing the data coming from the *Receiver.* A real-time clock is used to select the FIFO

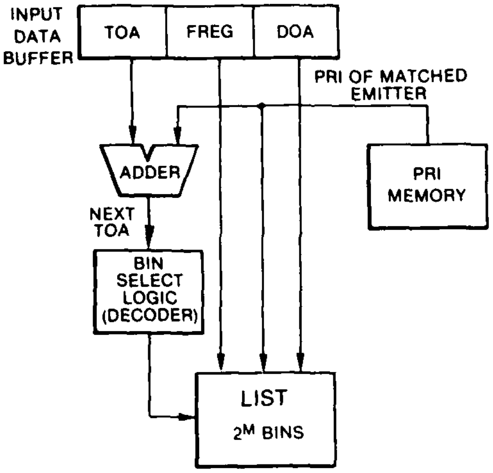


Fig. 4 — Loading the LIST after a match in the CAM

buffer from which data are loaded into the CAM. A time slot, using the same values of M and N derived earlier is generated from the real-time clock and used to determine the module number (a FIFO buffer) from which data are loaded into the CAM.

This procedure of loading the CAM attempts to make efficient use of the limited CAM space by only loading those emitters into the CAM which the system expects to see during the next period of time. Emitters are loaded into the CAM during the same time slot as their next expected arrival time. The size of this time slot is determined by the parameter N (the number of bits shifted). Therefore, the loading of an emitter into the CAM precedes its actual arrival by an amount of time less than or equal to the size of the time slot minus the delay in the LIST.

When loading the CAM from the LIST, an indentical window is applied on the real-time clock to determine the Bin for loading as was used on the NTOA word to determine the Bin for loading the LIST. Once the Bin is determined, data are unloaded sequentially from top to bottom (i.e., FIFO).

PERFORMANCE MEASURES

There are different measures which could be used to evaluate the performance of the signal sorter. An important measure is the delay an emitter encounters from the time its first pulse is received until it is completely identified and its parameters stored in the system file. A major objective of the system is to minimize this delay which allows it to handle a reasonably dense environment in real time. When a relatively small number of microprocessors are used in the processor array, the identification and filing functions cannot be realized without some filtering of the incoming data by the Associative Processor. Therefore the amount of filtering done by the Associative Processor has a great effect on the performance of the system. Thus, the ratio of the number of matched pulses to the number of unmatched pulses in the Associative Processor (CAM) represents an indirect measure of the system performance. Since the CAM unmatched data are passed into the microprocessor array for identification, the maximum size or number of pulses in the buffer in front of this array of microprocessors is also a reasonable performance measure. A larger number of the emitters waiting in this buffer would result in a longer delay in completing their processing. Therefore, performance measures may be chosen from the following:

1. The ratio of CAM MATCH/NO MATCH.

2. The maximum number of emitters waiting in the FIFO buffer of the array processors.

3. The maximum delay in processing an emitter.

4. The maximum number of emitters in any Bin of the LIST. This shows a lower bound for the required CAM size.

5. The maximum number of emitters waiting in the receiver output buffer (data to be processed by the Associative Processor). This shows whether the Associative Processing speed is adequate to process the incoming data rate.

Since the above performance measures are not independent, a subset is sufficient to measure the system performance. The simulation model measures all the above parameters, but only the size of the Processor Array buffer and the CAM Match/No Match ratio are shown in the tables of this report. Performance measures other than those listed above may also exist.

ARCHITECTURE TUNING

The exact values of emitter parameters to be encountered by the signal sorter are not always known. The environment simulation was based on generating emitter parameters which are random with a uniform distribution but bounded by some arbitrary upper and lower limits. These limits were set to represent previously observed values. Having a fixed system architecture for the signal sorter would limit its effectiveness to a limited range of certain emitter parameters, such as DOA and PRI distribution. In order to expand the range of environment data for which the sorter can operate properly, dynamic adjustments in the sorter architecture are necessary. Some particular parameters that affect the performance of the signal sorter considerably are the environment density (i.e., the number of emitters seen at the same time), the total number of pulses per second, the distribution of the PRIs, and the geometry of the emitters relative to the sorter (i.e., emitters distribution among the DOA cells). An exact knowledge of the values of these parameters would make decisions regarding hardware and software design much easier. Variations in the emitter parameters, incomplete data due to errors or corrupted data, or lack of information about the characteristics of new emitters in the environment could degrade the overall performance of a static (fixed) architecture of the signal sorter system. This assertion has been tested and shown to be true using the simulation models. A more flexible design, in which the hardware can be reconfigured and/or the software modified based on the characteristics of the environment data, would yield superior performance over a wider range of input data characteristics.

As stated earlier, the range of the emitter's PRI has a considerable effect on the structure of the LIST which is used for loading the CAM with the proper data of the predicted next arrival pulses. Keeping the total size of the LIST fixed, the structure of the LIST in terms of the number of modules and the size of each module could be based on the minimum and the maximum PRI. (See the section on implementation.)

Another example of an environment parameter that influences system design is the distribution of emitters in the DOA cells. A flexible and more effective design should allow the microprocessor array architecture to adapt itself to the above distribution. This self adjustment could be achieved by allocating more processing power to the more heavily populated DOA cells of the environment.

The adjustment of the architecture based on actual values of environment data could result in considerable improvement of the system performance. These flexibilities in the signal sorter design are referred to as architecture tuning.

This portion of the research work is associated with the architecture tuning of the Associative Processor section but the primary portion affected is the LIST section of the signal sorter. A detailed analysis and simulation of the effect of the distribution of PRI's on the configuration of the LIST is considered. First, the configuration of the LIST in terms of the number of modules and the module size are analyzed in relation to the PRI range and distribution. Other architectural concepts considered are the loading and unloading strategy of these LIST buffers as last-in-first-out (LIFO) or first-in-first-out (FIFO). A trade-off between using FIFO buffers and random access memory (RAM) modules for constructing the LIST is also analyzed.

FIFO buffers are memories which have built-in pointers to the next location for loading and updating automatically with every load or unload. Unloading is always done from the bottom of the buffer (first-in). The load/unload strategy is fixed by the buffer structure which is first-in-first-out. A major drawback of using FIFO buffers is the number of integrated circuit packages required (chip-count). When large numbers of LIST modules are required, a large number of FIFO integrated chips are required, since FIFO storage densities are much lower than conventional RAM memories.

Using a RAM memory to implement the LIST reduces the chip-count. One large integrated circuit (IC) memory could be divided into several blocks corresponding to the modules needed for the LIST. A number of external pointers are needed to manage the loading and unloading of these LIST modules. A FIFO strategy would require two pointers per module, while a LIFO policy would require one pointer per module.

Analysis and computer simulation of the LIST architecture and load/unload policies stated above, are presented in the following sections.

LIST Reconfiguration

The LIST is used as a buffer from which emitters are loaded into the CAM. It is an ordered list according to the next expected arrival times of the emitters in the environment. As shown in the section on the LIST Forming Processor, the ordering, which is based on the next time of arrival (NTOA), is not precise, but instead is based on a time *slot or window*, with all the emitters whose NTOA falls within the same slot loaded into the same module of the LIST.

It is also shown that the time slot selection was based on the range of emitters PRI distribution. Assuming a uniform distribution of the PRI's with some values for minimum PRI and maximum PRI, the time window depends on both the minimum and the maximum PRI.

The time window is selected as a group of bits in the middle of the NTOA word for loading the LIST. Similar bits from the real-time clock are used for loading the CAM from the LIST.

Two operations have to be performed on the calculated time word to extract the loading time window. First the N right most bits are dropped (shifted right). The next M bits constitute the time window, and the left most bits are ignored. The extracted M bits represent the module number (BIN #) associated with the emitters either for loading into the LIST or the CAM. Both N and M should be chosen based on the maximum and the minimum value of the PRI's. As an example, assume a sixteen bit time word with the binary value of 0011001011010110 and N = 7, M = 4. To compute the bin number the least significant N bits are dropped leaving 001100101. The next M bits give the bin number which is 0101. Therefore the bin for this time word is BIN 5.

The LSB of the real-time clock, the TOAs, and the PRIs has a value of 1 μs in the current model. The selection of the values of M and N should satisfy these conditions:

$$(1) \quad 2^N \leqslant \text{Min. PRI (in } \mu s)$$

$$(2) \quad 2^{N+M} \geqslant \text{Max. PRI (in } \mu s)$$

The first condition is necessary to provide a good distribution of the emitters over the number of BINS in the LIST. It means that the time slot will be set such that the same emitter will not be loaded twice in the same time slot Bin. The second condition is also necessary in order to cover all the range of NTOA including the emitters with the maximum PRI. That is, the time it takes for the system to cycle through all the bins should be greater than the maximum PRI.

Since the resulting M corresponds to the BIN number (or a module of the LIST) the upper bound on M should be kept to the minimum value which satisfies conditions (1) and (2). As an example consider the following system parameters:

Min. PRI = 200 μs, Max. PRI = 3000 μs.

The number of BINS can only vary between eight and 128 i.e., (M = 3 to 7). This would be a system hardware constraint.

Application of the conditions (1) and (2) results in the following (N and M can take on only integer values):

$$(1) \quad 2^N \leqslant 200, \quad \text{i.e., } N \leqslant 7$$

$$(2) \quad 2^{N+M} \geqslant 3000, \quad \text{i.e., } N + M \geqslant 12$$

The (N,M) pairs that satisfy the above conditions with the minimum values of M (using the equality in (2)) are (7,5), (6,6), (5,7). One of the above pairs should result in the best performance.

The above analysis determines the number of modules (M) required in the LIST and the position of the time window for determining the module number of the LIST for loading an emitter with a given next arrival time (NTOA). Assuming a fixed size for the LIST, in terms of total capacity in words, the number of words per module will vary with the value of M. As an example, if the system has a total of 64 basic FIFO units with eight words per unit, then the module size would vary between eight and 32 words as the number of modules vary between 64 and 16 respectively. This means the LIST could be configured as either 64 modules (bins) of eight words each, 32 modules of 16 words each, or 16 modules of 32 words each. These different configurations could be achieved with the same hardware by changing the interconnection scheme.

The total number of words in the LIST should be a function of the environment density, which in turn is a function of both the expected maximum number of emitters and the PRI distribution. Given a fixed size of the LIST, its configuration in terms of the number of modules should be a function of the PRI range, as the PRI range varies, the LIST should be reconfigured based on the above analysis.

Any LIST configuration has to satisfy both conditions (1) and (2). Condition (1) determines the upper limit on the bits to be shifted (N). Condition (2) determines the lower limit on (N + M). Hence the combination leaves the upper limit on M as infinite. Hardware costs and implementation factors should restrict the upper limit on M. Also, since larger values of M could result in a higher implementation cost, the inequality in condition (2) could be replaced by an equality. With these added restrictions, the conditions for the LIST configuration can be restated as follows:

(i) $M \leqslant M_{max}$ (Log_2 of the maximum number of modules allowed),

(ii) $N \leqslant [\mathrm{Log}_2\ PRI_L]$,

(iii) $N + M = [\mathrm{Log}_2\ PRI_H] + 1$ if $\mathrm{Log}_2\ PRI_H \neq$ integer

$= [\mathrm{Log}_2\ PRI_H]$ if $\mathrm{Log}_2\ PRI_H =$ integer,

where [ ] refers to the integer value of the quantity inside the brackets.

Let us assume the maximum number of modules is (128), therefore $M_{max} = 7$. Table 1 illustrates the result of applying conditions (ii), (iii) to several arbitrary PRI ranges (PRI_H, PRI_L). The list configuration which yields the best performance corresponds to an (M,N) pair from those given in the table.

Simulation runs for various values of M, N, PRI_L and PRI_H are shown in Table 2. Points other than those satisfying Eqs. (i), (ii), (iii) are also tested. In each case the optimal operating point is found to be from the set determined by the equations.

Table 1 — LIST Configuration as a Function of PRI_H, PRI_L

PRI_H, PRI_L	N + M	M_{MAX}	N_{MAX}	M,N (possible values)
2000,300	11	7	8	(4,7), (5,6), (6,5), (7,4)
2048,128	12	7	7	(6,6), (7,5)
3000,128	12	7	7	(6,6), (7,5)
4000,640	12	7	9	(4,8), (5,7), (6,6), (7,5)
4000,896	12	7	9	(4,8), (5,7), (6,6), (7,5)

Table 2 — LIST Configuration as a Function of PRI Distribution
(A/B = # NO MATCH IN CAM/ARRAY BUFFER SIZE)

Minimum PRI (PRI_L)	Maximum PRI (PRI_H)															
	2000				2048								2500			
128	6,6	5,6	6,5		4,7	5,6	6,5	7,4	4,8	5,7	6,6	7,5	5,7	6,6	7,5	5,6
	48/6	40/4	45/5		190/4	63/4	73/6	O.V.		155/4	56/4	71/6	66/3	25/4	41/4	352/4
256	6,6	5,6	6,5	5,7	4,7	5,6	6,5	7,4	4,8	5,7	6,6	7,5	5,7	6,6	7,5	5,6
	27/4	29/4	73/4	69/5	101/3	37/3	68/4						32/4	23/4		371/5
384	4,7	5,7	6,6	5,6	4,7	5,6	6,5	7,4	4,8	5,7	6,6	7,5	5,7	6,6		
	40/3	32/3	22/3	22/3	67/2	37/3	87/3	439/6					12/5	12/5		
512	4,7	5,7	6,6	5,6	4,7	5,6	6,5	7,4	4,8	5,7	6,6	7,5	5,7	6,7		
	26/3	18/3	22/3	22/3	68/3	20/3	85/3	401/4					10/4	10/4		
640	6,6	5,6	5,7		4,7	5,6	6,5	7,4	4,8	5,7	6,6	7,5	5,7	6,6	4,8	
	16/4	16/4	16/4		72/4	29/4	88/4						13/3	19/3	44/3	
768	6,6	5,6	6,5	5,7	4,7	5,6	6,5	7,4	4,8	5,7	6,6	7,5	5,7	6,6		5,6
	17/4	18/4	112/4	20/4		30/4	112/4						9/4	20/4		444/5
896	6,6	5,6	5,7		4,7	5,6	6,5	7,4	4,8	5,7	6,6		5,7	6,6	4,8	5,6
	23/3	23/3	11/3		70/3	36/3							15/4	17/4	15/6	476/6
1024	6,6	5,6	5,7		4,7	5,6	6,5	7,4	5,6	5,7	6,6					
	15/3	11/3	11/3													

Table continues.

Table 2 — LIST Configuration as a Function of PRI Distribution (Continued)
(A/B = # NO MATCH IN CAM/ARRAY BUFFER SIZE)

Minimum PRI (PRI_L)	Maximum PRI (PRI_H)															
	3072				4000				4096							
128	4,8	5,7	6,6	7,5	7,6	6,6	6,7	5,7	4,9	5,8	6,7	7,6	4,8	5,7	6,6	7,5
	336/6	37/3	14/3	41/3	17/4	17/4	17/5	15/4	X	62/4	13/4	17/4		15/4	18/4	57/4
256	4,8	5,7	6,6	7,5	5,7	6,7	6,6			5,8	6,7	7,6	4,8	5,7	6,6	
	84/4	19/4	21/4	81/5	12/5	12/5	11/5		X	33/6	12/6	22/6	47/6	17/6	23/6	X
384	4,8	5,7	6,6	7,5			6,7	5,7		5,8	6,7	7,6	4,8	5,7	6,6	7,5
	34/3	13/3	14/3	24/4			12/5	12/5	X	12/4	8/4	27/4	29/4	10/4	16/4	88/4
512	4,8	5,7	6,6	7,5			6,7	5,7	4,9	5,8	6,7	7,6	4,8	5,7	6,6	
	40/4	16/4	20/4	100/4			10/4	10/4	204/4	14/4	10/4	19/4	32/4	12/4	18/4	X
640	4,8	5,7	6,6	7,5	5,8	6,6	6,7	5,7	4,9	5,8	6,7	7,6	4,8	5,7	6,6	7,5
	19/3	12/3	16/3	79/3	11/4	14/4	9/4	9/4	110/3	15/3	10/3	25/3	34/3	11/3	25/3	114/3
768	4,8	5,7	6,6		4,8	6,6	6,7	5,7								
	12/3	8/3	25/3		18/3	15/3	11/3	11/3	X	13/3	8/3	21/3	32/3	8/3	21/3	X
896	4,8	5,7	6,6		4,8	5,7	6,6									
	18/3	9/3	24/3		16/3	13/3	31/3		X	13/4	8/4	X	34/4	13/4	X	X
1024	6,7	5,7	6,6						4,9	5,8	6,7	7,6	4,8	5,7	6,6	
	7/4	7/4	21/4						21/4	8/4	10/4	X	X	11/4	X	X

Table continues.

Table 2 — LIST Configuration as a Function of PRI Distribution (Concluded)
(A/B = # NO MATCH IN CAM/ARRAY BUFFER SIZE)

Minimum PRI (PRI_L)	Maximum PRI (PRI_H)										
	5118				7168				8000		
128	4,9	5,8	6,7	8,6	4,9	5,8	6,7	8,6	5,8	6,7	7,6
	X	20/3	7/3	9/3	X	14/4	12/4	15/4	11/3	9/3	17/3
256	4,9	5,8	6,7	8,6		5,8	6,7	8,6			
	364/4	12/4	10/4	15/4	X	8/3	9/3	18/3	X	X	X
384		5,8	6,7	8,6		5,8	6,7	8,6			
	X	14/4	8/4	24/4	X	6/3	7/3	22/6	X	X	X
512		5,8	6,7	8,6		5,8	6,7	8,6			
	X	10/4	8/4	22/4	X	7/4	7/4	21/4	X	X	X
640		5,8	6,7	8,6		5,8	6,7				
	41/5	11/4	10/4	21/5	X	6/3	6/4	X	X	X	X
768						5,8	6,7		5,8	5,9	4,9
	X	7/4	7/4	X	X	7/3	7/3	X	4/5	8/5	493/6
896	4,7	5,8	6,7		4,9	5,8	6,7				
	12/5	7/5	7/5	X	8/4	8/4	10/4	X	X	X	X
1024	4,9	5,8	6,7		4,9	5,8	6,7	5,7			
	27/5	6,5	7/5	X	6/3	6,3	8/3	431/5	X	X	X

The simulation runs which were used to obtain the results in the table held the following environment characteristics constant:

Number of emitters = 100

The maximum on time separation = 0.05 s

(This is the maximum length of time between the first pulse seen from each emitter in the scenario.)

Simulation run time = 0.1 s

The minimum and the maximum PRIs shown in the table are in microseconds. The entries in the table are the counts of NO MATCH COUNT/ARRAY PROCESSOR INPUT BUFFER SIZE for each run. The definition of these performance measures was in the previous section. Lower values indicate better performance. The top of each entry is the pair (M,N) *used for the LIST* configuration. Blank entries represent configurations which were not tried in this simulation. These configurations were not tried because adjacent points indicate that they would result in poorer performance.

Table 3 is a summary of the configurations giving the best results for the (PRI_H, PRI_L) pairs tested. A reasonable number of PRI pairs were tested. A value in the table applies for the range of PRI's starting with the row or column header and extending to the next higher PRI value. For example, the optimal operating point (6,6) shown for the (2048,256) PRI pair is valid for PRI_H from 2048 up to 2500 and PRI_L from 256 to 384.

The results in Table 3 are plotted in Fig. 5 to illustrate the optimal configuration for a given range of PRI pairs. It is seen from the figure that the three constraints (i), (ii), (iii) are satisfied. The value of M chosen was the minimum value which would satisfy (iii).

Table 3 — Summary of Best LIST Configuration vs PRI Distribution
(M,N = Log_2 (number of Bins, relative Bin size)

Minimum PRI (PRI_L)	Maximum PRI (PRI_H)								
	2000	2048	2500	3072	4000	4096	5118	7168	8000
128	5,6	6,6	6,6	6,6	6,6	6,7	6,7	6,7	6,7
256	5,6	6,6	6,6	5,7	6,7	6,7	6,7	5,8	5,8
384	5,6	6,6	6,6	5,7	5,7	6,7	6,7	5,8	5,8
512	5,6	6,6	6,6	5,7	5,7	6,7	6,7	5,8	5,8
640	5,6	5,7	5,7	5,7	5,7	6,7	6,7	5,8	5,8
768	5,6	5,7	5,7	5,7	5,7	6,7	5,8	5,8	5,8
896	5,7	5,7	5,7	5,7	5,7	6,7	5,8	4,9	4,9
1024	5,7	5,7	5,7	5,7	5,7	5,8	5,8	4,9	4,9

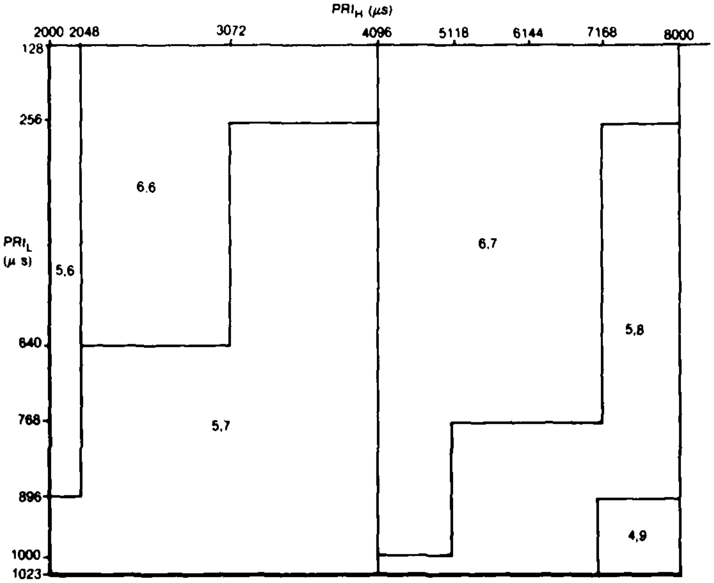


Fig. 5 — (M,N) LIST configuration vs PRI distribution

Implementation

The LIST consists of a number of LSI integrated circuit modules with changeable logical interconnection between them. Even though the hardware connections between the different circuits are fixed, the logical connections are under microprogram control. The logical connection and hence the configuration of the LIST into a certain number of independent first-in-first-out buffers is determined by the software executed by the system itself during actual operation.

The initial values of M,N are chosen as some "nice" values as determined by simulation. Reconfiguration of the LIST is accomplished by monitoring the maximum and minimum PRIs of the current emitters in the environment scenario. The microprocessor array keeps track of the PRIs and whenever a sizable change in either the PRI_H or PRI_L occurs, it outputs new values to the LIST control hardware which would result in reconfiguration of the LIST. The necessary amount of change in PRI_H or PRI_L to initiate a reconfiguration was left as a user input in the simulation. In the runs described in this report, all changes in PRI_H or PRI_L of any size caused reconfiguration. This is the worst case condition in terms of processing overhead. Any actual implementation would probably use some otner threshold.

Implementation of the LIST using two different types of storage modules is considered in this section. In the first implementation, hardware "FIFO" circuits are used, while in the second implementation, the LIST is constructed by using a RAM memory with a set of pointers.

FIFO Buffers Implementation

In this section a LIST architecture using FIFO buffers is presented. The LIST consists of a fixed number (K) of "FIFO" integrated circuits (ICs) of a fixed size and configuration logic. The array processor passes to the LIST configuration logic the maximum and the minimum PRIs. The LIST configuration logic performs the following functions:

(1) It maps the PRI pair (PRI_H, PRI_L) into the (M,N) pair based on a presorted table. The contents of the table are derived based on the simulation results shown earlier. Figure 6 shows the logic for the mapping of (PRI_H, PRI_L) into a (M,N) pair.

In the example shown in Fig. 6, it is assumed that the shifts are limited between 6 and 9 (N = 6 to 9). The number of modules is limited between 16 and 64 (i.e., M = 4,5 or 6). These values are those resulting from the simulation runs presented earlier and hence taken as an example for the hardware design. Considerations of the limits on M and N allow the optimization of the size of the mapping ROM table in terms of its word length. In this example, the ROM word length is limited to 4 bits. Two bits are used to encode the value M and the other two encode the value of N.

(2) Given an (M,N) pair generated by the mapping ROM, the LIST architecture is reconfigured accordingly. The configuration hardware generates the signals for the interconnection of the FIFO ICs such that they constitute 2^M modules with each module consisting of one or more ICs. Whenever a module contains more than one IC a logical path is created between the ICs within the module to increase the depth of that module. That is, if the number of modules is halved, the number of words per module is doubled.

As an example let us consider a LIST consisting of (16) FIFO modules and a PRI distribution that requires the LIST architecture to vary between (16) modules and (8) modules. Figure 7 shows the LIST with the interconnection logic. Note that when the LIST is configured as eight modules, the output of the even bins (e.g., BINØ) is routed to the input of the next bin to double the size of the modules.

(3) The (M,N) pair is also used by the LIST configuration logic to provide for the proper decoding of the emitters' NTOA and the real-time clock in order to load and unload the LIST accordingly. This configuration is achieved by ignoring the low order N bits of the time fields, then decoding the adjacent higher order M bits into a module number. The module number generated determines the module for the load/unload of data.

To further illustrate the LIST configuration hardware, let us consider the environment used to generate the results in Table 2. This environment has a PRI_H of 8 ms. and a PRI_L of 128 μs. For

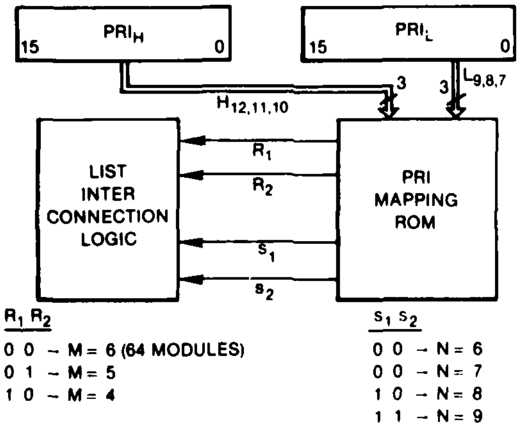
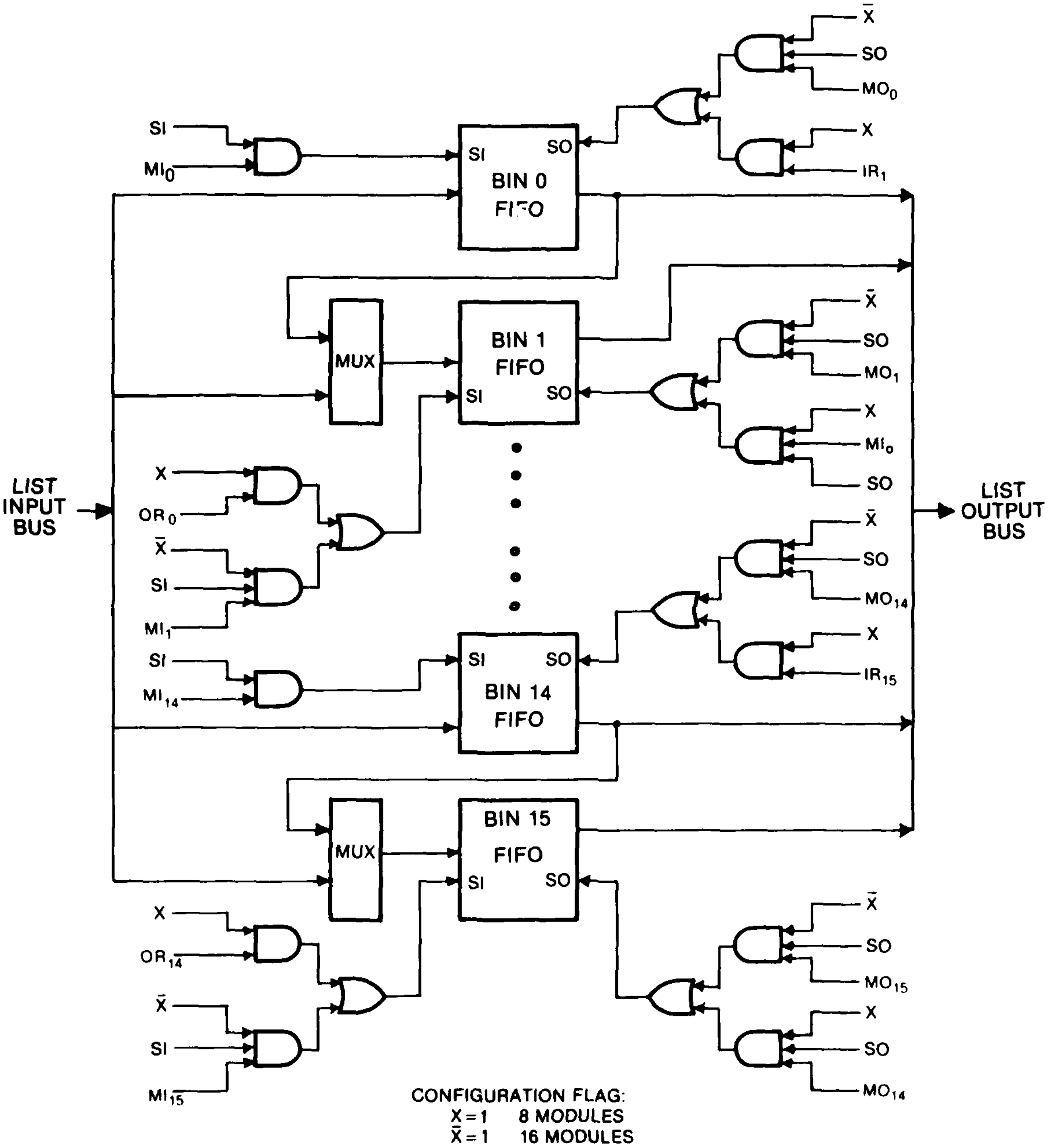


Fig. 6 — PRI's mapping ROM



CONFIGURATION FLAG:
X = 1 8 MODULES
X̄ = 1 16 MODULES

Fig. 7 — LIST interconnection

this particular environment, the minimum number of shifts (N) is equal to six (6). Also, the maximum value of (N + M) is equal to thirteen (13). Therefore given an emitter's NTOA, the higher order (3) bits and the lower order (6) bits should be ignored. The middle (7) bits are then mapped into the module number into which the emitter's data should be loaded. Figure 8 shows how the NTOA mapping could be done. This example illustrates that if the bounds of PRI_H and PRI_L are known, the size of the NTOA mapping ROM can be reduced. Similar procedure and hardware applies to mapping the real time clock for loading the CAM from the LIST.

Using a RAM for the LIST

The reconfiguration using FIFO buffers to form the LIST requires a few microseconds to tune the system to the new environment (different PRI's range). Its drawback is the large amount of hardware required to construct the LIST. Current technologies using FIFO buffers require a relatively large number of IC packages to implement.

An alternative design for constructing the LIST is to use a RAM memory to replace the FIFO buffers. In order to provide a modular buffer architecture with the RAM, added overhead is introduced associated with keeping and updating pointers to the sections of the RAM which simulate the relatively large number of logically independent buffer modules. Some of this added overhead occurs with each data word loaded into the RAM LIST or taken out of the LIST and loaded into the CAM.

In order to simulate the FIFO buffer modules by a RAM, two pointers are needed with each RAM section representing a module. One pointer, the LIST load pointer, keeps track of the top of the stack (last-in), while the LIST unload pointer points to the bottom of the stack (first-in). The load pointer gets incremented with each load into the RAM section (LIST module). The unload pointer gets incremented with each load from the LIST module to the CAM.

The total number of pointers to the LIST is thus twice the number of modules in the RAM LIST. Hence the number of pointers could be large for a system configuration that requires a relatively large number of buffer modules. A practical implementation then requires that memory locations within the RAM be used as pointers rather than using separate register pointers outside the RAM memory.

Another overhead caused by the RAM implementation is that which is associated with the reconfiguration process itself. As described earlier, whenever a significant change in the PRI_H, and/or PRI_L occurs, the system is tuned to the changed environment by selecting the number of buffer modules and the amount of shift which results in the optimal time window configuration.

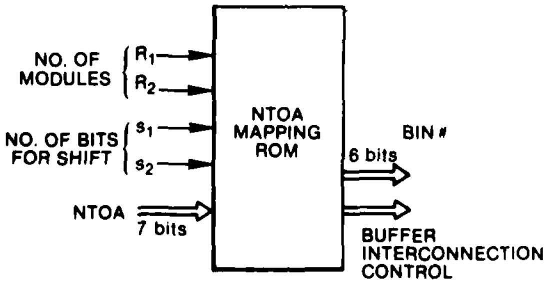


Fig. 8 — Next TOA (NTOA) mapping ROMs

In practice, PRI_H and PRI_L should not change frequently within a given scenario, and hence the architecture tuning is only needed to tune the system to one of a small number of different configurations. Therefore, the overhead associated with tuning the LIST configuration to the environment should occur at widely spaced times of significant change in the emitter environment (system turn-on, for example) and not on a continuous basis during the entire engagement.

Therefore, the aim of architecture tuning of the LIST in this case is simply to configure the LIST, based on PRI_H and PRI_L as part of the system initialization. The above statement assumes that reasonably accurate values of PRI_H and PRI_L are known in advance. When PRI_L and PRI_H are not known in advance, the system could be initially configured based on some practical predicted values of known PRIs and the system could tune itself later based on the PRI data collected by the system.

The procedure for reconfiguration then requires that the array processors keep track of the PRI_H and PRI_L values of the emitters identified and the number of emitters identified. After a reasonable number of emitters is identified, the PRI_H and PRI_L are considered to be valid for LIST configuration. The number to use would have to be experimentally determined. In the results reported here, the reconfiguration was done after each new emitter in the scenario was identified. This method gives the worst case in terms of processing time required. The array processors continue to keep track of the PRI_H and PRI_L. Reconfiguration of the LIST could be done again, when a new value of PRI_H or PRI_L is different from the previous values by some threshold value.

The LIST reconfiguration is then initiated by an interrupt from the array processors to the Load CAM processor. Upon receiving an interrupt, the Load CAM processor reads the new configuration status (number of modules in the LIST) and then adjusts the buffer pointers accordingly.

When the LIST is implemented by a number of FIFO integrated circuit modules only a simple controller is needed to perform the LIST management (LIST Forming and Load CAM) tasks. For a RAM LIST implementation, it is more desirable to use a microprocessor with internal data memory for the management of the LIST. The microprocessor capabilities are needed to provide for the address computations needed to access the LIST. The processor internal data memory is used to keep the pointers to the next available data word in the various sections of the LIST. Figure 9 illustrates the RAM LIST addressing hardware for loading an emitter data into the LIST and/or loading data from the LIST into the CAM. Part of the hardware is used to map the (PRI_H, PRI_L) pair into the configuration (i.e., the number of modules and the amount of shift), while the other part maps the NTOA/Real-Time clock into the module number for LIST loading/unloading.

SIMULATION RESULTS AND PERFORMANCE COMPARISON

Several implementations of the LIST have been simulated. Each implementation has the capability of LIST reconfiguration based on the PRIs range and distribution. A summary of the simulation results for these various implementations is presented in this section.

The environment characteristics in terms of number of emitters and PRI distribution were kept the same for most of the simulation runs which were used to evaluate the various LIST architectures. Also, the processing times for the Associative and Array processors were kept fixed. Hence, in all the simulation results presented in this section, the environment density (number of emitters) and the processing speeds were fixed to the same values unless otherwise noted.

The performance measures which are used from the simulation runs are the *number of NO CAM MATCH*, and the *size* of the buffer in front of the *Array processors*. These criteria are the

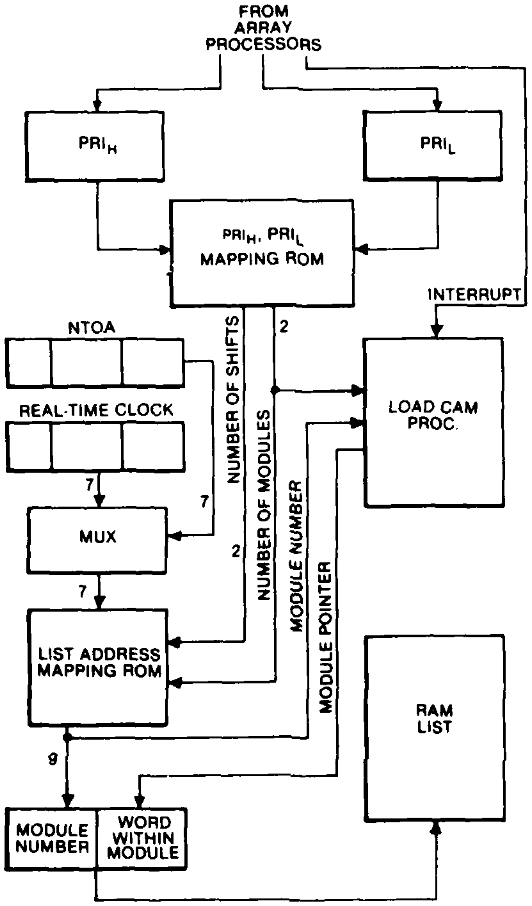


Fig. 9 — RAM LIST addressing

same performance measures as were used in Table 2. They are presented in the various tables as a pair of numbers in the above order, separated by a comma. Lower values of these parameters indicate better performance.

FIFO Implementation

In this implementation, physically separate FIFO buffers using FIFO integrated circuit modules are assumed to form the LIST. This architecture has the least amount of processing overhead in loading and unloading the LIST. The entire reconfiguration process is done in hardware, which eliminates any software overhead associated with reconfiguration. The first-in-first-out characteristics of the individual modules eliminates any overhead associated with the updating of pointers for proper loading and unloading of data into and out of the LIST respectively.

Table 4 shows the performance of static tuning versus that of dynamic tuning. For static tuning, the LIST configuration was fixed during the entire simulation run. That is the number of modules in the LIST, and the location of the time window were set in advance and remained fixed during the run. The performance measures shown in Table 4 for static tuning indicate clearly, the

Table 4 — Dynamic vs Static Tuning

Configuration M,N*	Static Tuning	Dynamic Tuning
4,9	28,3†	21,3
5,8	8,3	19,3
6,7	8,3	16,3
6,6	407,5	21,3
5,7	446,5	21,3

*M,N → NUMBER OF MODULES, TIME SLICE OF EACH MODULE

†28,3 → 28 CAM NO MATCHES, 3 WORDS MAX BUFFER SIZE

need for determining the configuration of the LIST based on the PRI distribution. For the emitter environment used, the performance of the static architecture varied widely for different configurations. But by using a dynamically tuned architecture, a consistant performance was achieved. Arbitrary selection of a static configuration could result in degraded performance. For example the configurations (N = 6, M = 6 or N = 5, M = 7) would not be good choices for this particular simulation environment as shown in the table.

In the dynamic tuning runs, the LIST was reconfigured dynamically based on the PRI$_H$ and PRI$_L$ encountered in the environment. The configuration values (N,M) in this case were used for initial configuration of the LIST. The LIST was then reconfigured based on the PRI$_H$ and PRI$_L$ encountered by the signal sorter. In the simulation model, the PRI$_H$ and PRI$_L$ are channeled into the LIST configuration directly through the hardware. The configuration hardware automatically allows the coupling or decoupling of modules. A change in the configuration from 32 buffer modules into 16 modules would result in ordering the entire data that are contained in the 32 modules into just the 16 modules with no loss of data. This is easy to implement due to the implicit pointer characteristics of the FIFO buffer circuit modules. The data in the BINØ would be combined automatically, with no added overhead, with the data in BIN 1 to form the new BINØ (see Fig. 7).

In a RAM LIST implementation, which will be discussed in the next section, data merging after reconfiguration would be time consuming because the pointers are not implicit. It could be more efficient just to ignore loading the data in the higher numbered modules into the CAM. For example, when reconfiguring from 16 to eight modules, just ignore the data currently in modules nine to 16. Table 5 shows the simulation results for FIFO LIST configuration which ignored the data in the higher modules when the configuration changed into a smaller number of modules. This case is called Dynamic Tuning (2) in the table. These results are included for comparison purposes only since there is no advantage of such an implementation using the FIFO ICs. As shown in the table, the performance of such an implementation is not as good as the previous case where no data in the LIST is lost, although the dynamic tuning again shows a consistent performance as compared to the performance of the various static configurations.

RAM Implementation

Previous results show that the FIFO buffer implementation requires a relatively large number of FIFO ICs. An alternative implementation of the LIST is to use RAM buffers for the LIST. A RAM buffer with a total capacity equal to that of the entire FIFO implementation would

Table 5 — Static vs Two Cases of Dynamic Tuning

Configuration M,N	Static Tuning	Dynamic Tuning (1)	Dynamic Tuning (2)
4,9	28,3	21,3	51,8
5,8	8,3	19,3	51,8
6,7	8,3	16,3	51,8
6,6	407,5	21,3	51,8
5,7	446,5	21,3	51,8

require a much smaller number of RAM integrated circuit packages. Thus this implementation reduces the total number of system components (ICs) considerably. The basic drawback is the need for, and the overhead associated with a large number of explicit pointers.

The RAM has to be divided into a number of sections corresponding to the number of independent modules in the LIST. Two implementations are considered for the RAM LIST. In one implementation, each section of the RAM is organized as a FIFO buffer. That is, the data are read out of each section of the memory in the same order as they were written. In this case, two pointers are needed for each section, one to point to the top of the stack, while the other keeps track of the location of the bottom of the stack. These pointers are referred to as the Load pointer and the Unload pointer respectively. When an element is added to the stack the Load pointer (bottom) is incremented to indicate the next available location. Data are moved into the CAM from the top of the stack. The Unload (top) pointer is also incremented with each unload operation. A stack is non-empty as long as the value of the bottom pointer is larger than the value of the top pointer. When the two pointers are equal the stack is empty.

In the second RAM implementation, each section of the RAM is organized as a last-in-first-out (LIFO) stack. This organization would reduce the number of pointers needed to one for each module of the LIST. This implementation would reverse the order of loading and unloading of the emitters within the same time slot. Therefore, this scheme presents a greater probability of loading data into the CAM after the pulse has arrived, hence wasting valuable space in the small CAM and causing extra NO MATCH conditions with input data which had already been identified. Each NO MATCH in the CAM increases the processing load of the microprocessor array, and thus decreases system throughput.

Associated with both RAM implementations is additional processing time overhead. The overhead has two distinct components. First, with each reconfiguration of the LIST, the Load CAM processor must know the maximum number of modules in the LIST. This information is needed to update the pointers. For example, a change in the configuration from 32 to 16 modules requires the adjustment of the pointers of the 16 modules which are no longer active to indicate that they are empty. The current data in these modules are ignored and not loaded into the CAM. To avoid this loss of data would require joining adjacent modules (compaction) to form 16 modules from the previous 32. This process would add an even greater amount of overhead which would greatly affect the real-time operation of the system. The simulation results in this section represent only cases where no compaction was made. The compaction of the LIST section has been simulated and the results did confirm that LIST compaction would degrade system performance.

The second component of the added overhead occurs with every load into and every unload from the LIST. Since there are one or two pointers associated with each memory section, the Load CAM processor must first identify the section in order to select the appropriate pointer(s). This action is accomplished by reading the module number for loading or unloading before the appropriate pointer is selected and used to address the LIST.

The processing overhead, discussed above, is taken into account in the simulation model by increasing the Load CAM processing time associated with each load into the LIST or load into the CAM. This increase is represented by a single constant.

Table 6 shows a summary of the simulation results for various implementations including the two RAM LIST organizations. Definitions of the difference in these configurations used in the simulation to obtain these results are given in Fig. 10. The environment density and the fixed processing times, in the various components of the signal sorter, were held constant for all the simulation runs except when otherwise noted. The only variations were in the LIST organization and the Load CAM processing time.

Two important aspects of the RAM LIST implementation deserve additional considerations. One aspect is the effect of the additional overhead on the overall signal sorter performance. To attempt to learn the impact of the overhead, the Load CAM processing time was varied to obtain the results in Table 6. Simulation results using Load CAM processing times of 1 μs and 2μs are shown. Results when using a 3 μs processing time are shown later. The second aspect is the ADVANCE LOAD TIME parameter. Thus far, we have considered the same time window for both loading and unloading the LIST. That is, emitters are loaded from the LIST to the CAM during the time slot during which their next pulse is expected to arrive. The simulation program allows for the addition of a bias to the real-time clock which provides an advance loading of data into the CAM. That is, the Load CAM processor tries to load emitter data into the CAM ahead of the time slot which contains its expected next arrival time. The advance loading might be of particular use in the case of the LIFO RAM implementation. Intuitively, the performance of the LIFO RAM should improve when advance loading is allowed. It is also clear that the total effect of an advance loading is also dependent on the CAM Load processing time. For example, if the CAM Load processing time is small, then advance loading would have a lesser effect on performance as compared to its effect when the CAM Load processing time is larger.

In order to verify the effect of the above two aspects (the CAM Load processing time and the ADVANCE LOAD TIME), simulation runs for various values of these parameters were performed.

Table 7 shows the effect of variation of the ADVANCE LOAD TIME on the performance of the FIFO LIST. Since the FIFO LIST does not have any overhead for loading or unloading

Table 6 — Summary of Simulation Results

Configuration M,N	Static Tuning Load Time 1 μs EVS7	Dynamic Tuning					
		Load Time 1 μs				Load Time = 2 μs	
		EVS2	EVS0	EVS3	EVS1	EVS1	EVS3
4,9	28,3	22,3	51,8	51,8	61,8	124,8	55,8
5,8	8,3	19,3	51,8	51,8	61,8	124,8	55,8
6,7	8,3	16,3	51,8	51,8	61,8	124,8	55,8
6,6	407,5	21,3	51,8	51,8	61,8	124,8	55,8
5,7	446,5	21,3	51,8	51,8	61,8	124,8	55,8

See Fig. 10 for description of program versions.

Program Name	List Implementation
EVS7	FIFO LIST, Fixed Window (Static Tuning)
EVS2	FIFO LIST, Dynamic Tuning
EVS0	The same as EVS2, except data in upper modules are lost when configuration changes to lower number of modules
EVS1	LIFO RAM LIST
EVS3	FIFO RAM LIST

Fig. 10 — Definition of simulation programs

Table 7 — Effect of Advance Load Time on FIFO LIST

Load Time in μs	Advance Load Time in μs				
	0	16	32	64	128
1	54,8	52,8	53,8	57,8	75,8

the LOAD TIME is fixed to 1 μs. For this particular environment and LIST configuration, increasing the ADVANCE LOAD TIME decreased performance.

Table 8 shows the effect of the LOAD TIME and the ADVANCE LOAD TIME on the performance of the FIFO RAM LIST. The same simulation parameters were repeated using the LIFO RAM LIST configuration and the results are shown in Table 9.

Comparison of Results

Referring to Table 6, the fixed configuration of the LIST has a better performance than the dynamic reconfiguration in those cases when the appropriate values of N,M are used. That is, the PRI_H and PRI_L values are known in advance. In order to achieve superior performance with a fixed configuration, an a priori knowledge of PRI_H and PRI_L is necessary and a static environment is needed.

Dynamic reconfiguration using FIFO LIST (EVSØ) has the same performance as that using FIFO RAM LIST (EVS3) assuming no LIST compaction is performed. FIFO LIST (EVS2) performs better than FIFO RAM (EVS3) when FIFO compaction is considered. The compaction is an intrinsic property of the FIFO LIST which does not cost any processing time overhead. Compaction of RAM LIST is not implemented because of the amount of extra processing overhead involved.

The FIFO RAM LIST (EVS3) gives a better performance than the LIFO RAM LIST (EVS1). In comparing the performance of the FIFO LIST with that of the RAM LIST, more LOAD CAM processing time should be considered for the latter to simulate the extra overhead involved with the RAM. For example, a FIFO LIST with 1 μs CAM LOAD TIME should be compared with a RAM LIST with 2 μs CAM LOAD TIME for an accurate performance comparison.

Table 8 — Effect of Load Time and Advance Load Time on FIFO RAM LIST

Load Time in μs	Advance Load Time in μs				
	0	16	32	64	128
1	51,8	52,8	53,8	57,8	75,8
2	55,8	51,8	53,8	57,8	76,8
3	62,8	53,8	53,8	56,8	75,8

Table 9 — Effect of Load Time and Advance Load Time on LIFO RAM LIST

Load Time in μs	Advance Load Time in μs				
	0	16	32	64	128
1	61,8	50,8	49,8	52,8	67,8
2	124,8	69.8	49,8	51,8	61,8
3	225,8	131,8	86,8	55,8	58,8

The effect of an ADVANCE LOAD TIME on performance is shown in Tables 7, 8, 9. Table 7 shows that increasing the ADVANCE LOAD TIME of the FIFO LIST architecture, eventually degrades performance. As expected, while the ADVANCE LOAD does not have any significant effect on a FIFO structure (Table 8), it affects the performance of the LIFO structure (Table 9). As shown in Table 9, increasing the ADVANCE LOAD TIME to 32 μs has a better effect on performance than decreasing the loading time from 2 μs to 1 μs.

It is also clear, from Table 9, that the effects of the LOAD CAM processing time and the ADVANCE LOAD TIME on the LIST performance are interdependent. That is, increasing the ADVANCE LOAD TIME results in much larger improvement in performance in the slower LOAD CAM processor case than when using the relatively faster processor.

CONCLUSIONS AND FUTURE RESEARCH

One of the critical stages of the signal sorter system is the Associative Processing stage which filters the incoming data stream. Greater filtering effect could be achieved by better prediction of the next arrival time (NTOA) resulting in loading the CAM with the appropriate emitter data at the right time. Without predicting the NTOA, the Associative Processor would require a CAM size equal to or greater than the number of emitters in the environment. This number is not known in advance and could easily be large enough such that a CAM of that size would be very difficult to implement. The look-ahead scheme simulated in this model of NTOA prediction and CAM loading reduces the required number of words in the CAM. The necessary CAM size is no longer tightly coupled to the number of emitters in the environment, it is a function of the PRI distribution and processing throughput of the system. In the simulation model, 24 CAM words have been used to handle up to 500 different emitters in the environment simultaneously. This CAM size is realizable on a 22.5 cm (9 in.) by 22.5 cm (9 in.) circuit board using commercially available parts, and has proved to be of sufficient size in this model.

The fixing of the small size of the CAM was achieved by adding to the system a hardware LIST to buffer and order the data before being loaded into the CAM. It is the architecture of this LIST buffer which was investigated in this research.

These simulation results show that architecture tuning (or dynamic reconfiguration) of the LIST could improve the performance of the signal sorter considerably. The FIFO LIST configuration resulted in the best performance, but would be expensive and difficult to implement due to the large number of integrated circuit packages that would be needed. An alternative implementation is the FIFO RAM. The simulation results indicate that the processing time overhead, associated with updating the pointers and doing the reconfiguration, does not degrade the performance significantly. Implementing the RAM LIST in the system would therefore be more cost effective, requiring less space and power, than adding a larger size CAM.

Dynamic tuning of the LIST should be utilized when the environment density and the PRI distribution are not known. Once enough intelligence data on the environment has been collected and the PRI distribution is known, the appropriate fixed configuration of the LIST should be programmed into the system. The simulation results have indicated that better performance is obtained when the appropriate LIST configuration is kept fixed. But the simulations have also shown that fixing the LIST structure (static architecture) performs better only when the environment is well known beforehand. When the environment is not well known in advance, or unusual new emitters are encountered, the simulations show that the static system can saturate much quicker than the dynamic architecture system.

Relatively dense environments have been simulated, in order to investigate the effectiveness of the small CAM combined with the FIFO LIST. The simulations show that in a dense environment, the throughput of the array processors, the CAM size, and the LIST management processor speeds become limiting factors. A summary of the simulation results using dense environments are shown in Table 10. The simulation program (EVS4) had a FIFO LIST architecture like EVS2. The emitter turn-on times were in groups of 100 at 0.1-s intervals. The number of emitters was increased up to 500 emitters which is equivalent to approximately one-half million pulses/second. The CAM and the LIST were still fairly effective in filtering the known data. In order to handle the above data rate, the CAM size had to be increased and the microprocessor array microcycle time

Table 10 — Performance in a Dense Environment

Number of Emitters	Array Processor Microcycle Time (μs)	CAM Size	Performance No Match/Array Buffer Size
100	0.3	24	131/5
200	0.3	24	1099/8
300	0.3	24	Array Buffer Overflow
300	0.05	24	3326/17
500	0.05	24	Array Buffer Overflow
500	0.1	48	Array Buffer Overflow
500	0.05	48	20566/17
500	0.03	48	15268/17

Run time = 1 s.

had to be dropped from 300 ns to 50 ns. With current technology, a microcycle time of 100 s is realizable. The additional drop in the microcycle time (from 100 to 50 ns) could be achieved by increasing the number of processors (e.g., 6 processors instead of 3) in the array and/or modification in the processor array architecture.

In order to utilize the signal sorter for different and varying environments, more architecture tuning research is needed. For example, the research presented in this report shows that more processors in the array are required to process larger density environments in real-time. The current fixed task allocation could be a costly approach to handle very dense environments; the use of independent processors and a dynamic allocation would be more practical and cost effective. In the existing system, three processors are utilized in the array. These processors are preassigned to handle the emitters based on a fixed DOA distribution. The design lacks flexibility and could be inadequate if the emitters tend to concentrate in a few DOA cells and are not uniformily distributed over the entire range. This could cause saturations in some sections of the internal memory of the processors, since the memory is also allocated based on a uniform distribution of DOAs. Dynamic allocations of both memory space and physical processors could lead to better utilization of the system hardware and consequently result in better performance.

More simulations should be performed on the LIST structures in dense environments. Also needed is an analysis of the effect of exotic emitters, such as frequency agile emitters, on the system performance. It needs to be determined if the circuit architecture design can handle these problems, and if not, what modifications need to be made in the architecture. Work on these problems is currently being performed, and will be documented in future reports.

Appendix

ENVIRONMENT GENERATION SUBROUTINES

```
      SUBROUTINE NDAGE
C*****************************************************************
C
C  KEY VARIABLES:
C
C     EMS -    SEPARATION OF THE INITIAL TURN-ON TIMES OF
C              THE EMITTERS
C     E(I,J) - PARAMETER I OF EMITTER J, REFER TO FIGURE 1
C              FOR DEFINITION OF EACH PARAMETER
C
C******************************************************************
      IMPLICIT INTEGER*4 (I-N)
      COMMON/EMIT/E(100,25),EMDUM(5),LLLL,MMMM,ENDUM(102)
      COMMON/RAND/IRAN,JRAN
      COMMON/DA/EMS
C   THIS ROUTINE ESTABLISHES INITIAL VALUES FOR ALL EMITTERS
C   INITALIZE VELOCITIES AND TYPE TO 0
C START
      DO 15 J=1,100
      E(J,20)=0.
      E(J,21)=0.
      E(J,22)=0.
 15   E(J,23)=0.
C   INITALIZE VELOCITIES OF MOVING EMITTERS
      DO 25 J=1,100,10
      E(J,23)=-1.
      E(J,20)=RND(IRAN)*200.-100.
 25   E(J,21)=RND(IRAN)*200.-100.
C   SET TYPE FOR COLLISION EMITTERS
      DO 35 J=2,100,10
 35   E(J,23)=+1.
C  ENTER MAIN LOOP
      DO 100 I=1,100
C   SET XYZ COORDINATES
      E(I,1)=(RND(IRAN)*20.+2.)*10.**3
      E(I,2) =(RND(IRAN)*44.-22.)*10.**3
      E(I,18)=RND(IRAN)*1000.
C   SET EMITTER TRANSMITTER/ANTENN@ GAIN
      E(I,3)=RND(IRAN)*50.+30.
C   SET EMITTER ANTENNA BEAMWIDTH
      E(I,4)=RND(IRAN)*.41+.09
C   SET EMITTER ANTENNA SIDELOBE LOSS
      E(I,5)=RND(IRAN)*15.+15.
C   SET INITIAL ANTENNA ANGLE
      E(I,17)=ATAN2(E(I,2),E(I,1))
      IF(E(I,17).LT.0)E(I,17)=E(I,17)+6.2832
C   SET EMITTER MAX ANTENNA ANGLE
      E(I,6)=E(I,17)+1.5
C   SET EMITTER MIN ANTENNA ANGLE
      E(I,19)=E(I,17)-1.5
C   SET EMITTER ANTENNA SCAN RATE
 3    E(I,7)=RND(IRAN)*1.5+.5
```

```
C   SET EMITTER PRI
      E(I,8)=RND(IRAN)*.0017+.0003
C   SET EMITTER PULSE WIDTH
      E(I,9)=RND(IRAN)*.000003+.000001
C   MODIFY PULSE WIDTH FOR CW EMITTERS
C     IF(I.EQ.8)E(I,9)=.0001
C   SET EMITTER CARRIER FREQUENCY
      E(I,10)=RND(IRAN)*10.**10+2.*10.**9
C   SET EMITTER TURN ON AND OFF TIMES
      E(I,11)=RND(IRAN)*EMS
    8 E(I,12)=1.
C     INITALIZE EMITTER POWER SEEN AT PROCESSOR TO -500.
      E(I,13)=-500.
C   INITALIZE THE TOA TO THE ON TIME
      E(I,14)=E(I,11)
C   INITALIZE THE DOA AND THE ON FLAG TO 0
      E(I,15)=0.
      E(I,16)=0.
  100 CONTINUE
C   INITALIZE EMITTERS WITH PULSE GROUPS
C     DO 20 I=5,100,20
C     N=I+1
C     K=I+2
C     DO 21 J=1,25
C     E(N,J)=E(I,J)
C21   E(K,J)=E(I,J)
C     E(N,11)=E(I,11)+E(I,9)*2.
C     E(K,11)=E(N,11)+E(I,9)*2.
C     E(N,14)=E(N,11)
C     E(K,14)=E(K,11)
C20    CONTINUE
C  ELIMINATE SIDELOBES FOR TRACKING EMITTERS
      DO 30 K=3,100,20
   30 E(K,5)=0
      RETURN
      END

      SUBROUTINE NEMIT
C************************************************************************
C
C  KEY VARIABLES:
C
C     NE -     NO. OF EMITTERS IN THE ENVIRONMENT
C     TM -     CURRENT SYSTEM TIME
C     E(I,J) - PARAMETER I OF EMITTER J, REFER TO FIGURE 1
C              FOR DEFINITION OF EACH PARAMETER
C     XR,YR,ZR - X,Y,Z COORDINATES OF SIGNAL SORTER PLATFORM
C     DX,DY,DZ - X,Y,Z DISTANCE BETWEEN EMITTER AND SIGNAL
C                SORTER PLATFORM
C     R -      STRAIGHT LINE DISTANCE BETWEEN EMITTER AND
C              SORTER PLATFORM
C     SL -     SIDELOBE LOSS SEEN BY SIGNAL SORTER
C     ABE -    ANGLE BETWEEN CENTER LINE OF EMITTER AND
C              SIGNAL SORTER
C     CONST -  CONSTANT USED IN RANGE ATTENUATION EQUATION
C
C************************************************************************
      IMPLICIT INTEGER*4 (I-N)
      COMMON/EMIT/E(100,25),TM,XR,YR,ZR,CONST,NE,L,T(100),TMIN,TMS
      COMMON/EE/IMOD,N3,J3,ITLOC,ITPRI,EPT,NEE,TEEP,IPNT,IDAL(64),ILF
```

```
     1 ,FTIM,TEEL,TTLOC,TLEEP,IMODEL,IP(64),IFF,IHL(64),ILPNT,ILP
      COMMON/RAND/IRAN,JRAN
C     DOUBLE PRECISION SL,DX,DY,DZ,ABE,R,A
C START
      DO 3 J=1,NE
C   CHECK THE ON TIME AND THE OFF TIME
      IF(TM.GT.E(J,12))GO TO 3
      IF(TM-E(J,11)) 3,1,1
C   UPDATE THE ANTENNA ANGLE AND CHANGE SIGN IF LIMITS EXCEEDED
1     E(J,17)=E(J,17)+E(J,7)*E(J,8)
      IF(E(J,17).GE.E(J,6)) E(J,7)=-E(J,7)
      IF(E(J,17).LE.E(J,19)) E(J,7)=-E(J,7)
C   CALCULATE THE EMITTER POSITION
      E(J,8)=E(J,8)
      IF(E(J,23))9,10,11
C   CALCULATE FOR FIXED EMITTER
10    DX=E(J,1)-XR
      DY=E(J,2)-YR
      DZ=E(J,18)-ZR
      GO TO 50
C   CALCULATE FOR MOVING EMITTER
9     E(J,1)=E(J,1)+E(J,20)*E(J,8)
      E(J,2)=E(J,2)+E(J,21)*E(J,8)
      E(J,18)=E(J,18)+E(J,22)*E(J,8)
      GO TO 10
C   CALCULATE FOR COLLISION COURSE EMITTER
11    DX=E(J,1)-XR
      DY=E(J,2)-YR
      DZ=E(J,18)-ZR
      R=SQRT(DX*DX+DY*DY+DZ*DZ)
      E(J,20)=600.*DX/R
      E(J,21)=600.*DY/R
      E(J,22)=600.*DZ/R
      GO TO 9
C   CALCULATE THE DOA
50    E(J,15)=ATAN2(DY,DX)
      IF(E(J,15).LT.0)  E(J,15)=E(J,15)+6.2832
C   DETECTION OF THE SIDELOBE POWER
      SL=0.
      ABE=E(J,17)
      IF(ABE.GT.6.2832)ABE=ABE-6.2832
      IF(ABE.LT.0.)ABE=ABE+6.2832
      ABE=ABS(E(J,15)-ABE-3.1416)
      IF(ABE.LE.E(J,4)) GO TO 4
      SL=E(J,5)
C   CALCULATION FOR THE RECEIVER POWER
4     R=SQRT(DX*DX+DY*DY+DZ*DZ)
      E(J,13)=E(J,3)-CONST-20.*ALOG(R/1852.)-SL
C   CALCULATION FOR THE TOA
      E(J,14)=E(J,11)+R/(2.9979*10.**8)
C   UPDATE THE ON TIME
C     IF(J.EQ.9) E(J,8)=E(J,8)+RND(IRAN)*300.*10.**-6-RND(IRAN)
C    1 *150.*10.**-6
      E(J,11)=E(J,11)+E(J,8)
      IF(E(J,11).LE.E(J,14)) WRITE(1,20) J
20    FORMAT(2X,'EXCESSIVE PRI IN EMIT. J=',I3)
C    CALCULATION FOR FREQUENCY HOPPING EMITTERS
C     IF(J.LE.NEE) E(J,10)=E(J,10)*(0.8+0.4*RND(IRAN))
3     CONTINUE
60    RETURN
      END
```